HALCON

a product of MVTec

Quick Guide

HALCON 24.11 *Progress-Steady*

MVTec Software GmbH

Building Vision for Business

A quick access to the functionality of HALCON, Version 24.11.1.0

More information about HALCON can be found at: `http://www.halcon.com`

About This Manual

This manual introduces you to HALCON. It is intended for beginners without prior knowledge of HALCON.

This manual can also be used as a reference guide to several other HALCON manuals, as it interconnects them along the following topics:

1. Installing HALCON (page 7)
   This chapter introduces the MVTec Software Manager (SOM).
2. HALCON Architecture (page 9)
   Some theoretical background, needed to understand what HALCON is and how it works.
3. How to Develop Applications (page 15)
   This chapter explains three basic approaches for developing with HALCON and guides you through a first programming example.
4. How to Continue (page 19)
   This chapter refers to additional sources of information.

Contents

Chapter 1

Installing HALCON

For Linux und Windows users, we recommend downloading and installing HALCON via the MVTec Software Manager (SOM). SOM is an installation manager for software packages. It provides access to a remote catalog of products, and supports, among other features, downloading and installation of packages. A step by step introduction on how to install HALCON via SOM can be found in the Installation Guide.

Chapter 2

HALCON Architecture

HALCON's basic architecture is depicted in figure 2.1. The main part is the image processing library, which consists of more than 2000 operators. You can also develop your own operators in the form of so-called extension packages (page 11). You use the operators in your application via language interfaces (page 11) like HALCON/C++ or HALCON/Python. These are libraries which allow a direct use of the operators in the typical programming style of the different programming languages.

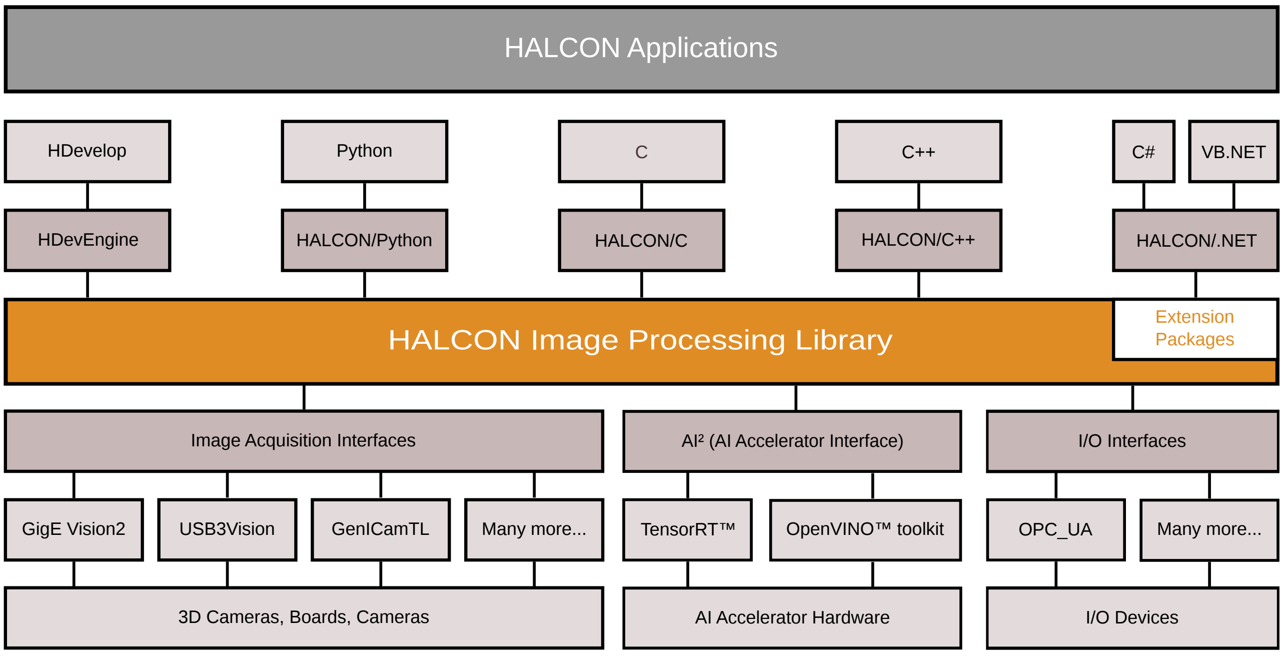


Figure 2.1: Basic architecture of HALCON.

For the access of image acquisition devices, HALCON provides image acquisition interfaces (page 13) which allow you to use quite different acquisition devices in a common way. The libraries containing the device-specific implementations are loaded dynamically when needed. In the same fashion, I/O devices are accessed through device-specific I/O interfaces (page 13).

2.1 Operators

Whenever any kind of functionality is used from the HALCON library, it is done via an *operator*. Most of them comprise multiple methods, which are selected via parameters. A full list of all operators can be found in the HALCON Operator Reference available in HDevelop, .NET, Python, C++, and C syntax. Important features of operators are:

- There is no hierarchy among operators. From the software architecture point of view, all operators are on the same level.
- Of course, there are logical groups of operators. For example, this can be seen by the classes offered for C++ and .NET, where operators processing the same data type are member functions of the corresponding classes.
- The design of the operators follows the rules of the open architecture. Therefore, you can create your own operators and thus extend HALCON (see section 2.2). The Extension Package Programmer's Manual contains detailed information about extending the operator library.
- Many operators can make transparent use of automatic parallelization, which allows an easy way of speeding up the program when using large images on a multi-processor or multi-core computer. Detailed information on parallel programming can be found in the technical note Parallel Programming, as well as in the Programmer's Guide.
- Operators have standardized rules for ordering input and output parameters: *input iconic, output iconic, input control, and output control* (see section 2.1.1). Not all of the groups might be needed for a given operator. In general, input parameters of operators are not modified, which results in a clear and simple semantics. There are only a few exceptions to this design, e.g., `set_grayval`, `overpaint_gray`, and `overpaint_region`.

2.1.1 Parameters and Data Structures

HALCON has two basic types of parameters: *iconic data* and *control data*. Images, regions, and XLDs (eXtented Line Description) belong to the iconic data.

Images consist mainly of channels, i.e., matrices containing pixel values. All channels of an image have the same size. For detailed information about pixels and channels, please read the chapter "Image" of the HALCON Operator Reference.

For each image, the so-called *region of interest (ROI)* specifies which part of the image is processed. The ROI can be defined very flexibly (from a simple rectangle to a set of unconnected pixels). For details about ROI handling see the Solution Guide I, Region Of Interest on page 25.

Regions are a set of pixels. The pixels of a region do not need to be connected. Even an arbitrary collection of pixels can be handled as a single region. With the operator `connection` a region can be split into its *connected regions*, i.e., components consisting of connected pixels.

XLDs comprise all contour and polygon based data. Subpixel-accurate operators like `edges_sub_pix` return the contours as XLD data. A contour is a sequence of 2D control points, which are connected by lines. Typically, the distance between control points is about one pixel. XLD objects contain, besides the control points, so-called local and global attributes. Typical examples for these are, e.g., the edge amplitude of a control point or the regression parameters of a contour segment. Besides the extraction of XLD objects, HALCON supports further processing. Examples for this are the selection of contours based on given feature ranges for the segmentation of a contour into lines, arcs, polygons or parallels.

The control data includes handles and basic data types like integer, real, string.

Handles are references to complex data structures, e.g., a connection to an image acquisition interface or a model for the shape-based matching. For efficiency and data security reasons, not the entire structure but only the handle is passed between the operators. Handles are magic values that must not be changed and can differ from execution to execution and version to version. They are automatically cleared once all references are overwritten. Examples where handles are used are graphics windows, files, sockets, image acquisition interfaces, OCR, OCV, measuring, and matching.

2.2 Extension Packages

HALCON may be extended by new operators. Although HALCON already contains an abundant set of operators for various tasks, you may wish to implement new operators, e.g., to access a special hardware or to implement an alternative algorithm. To do so, HALCON provides the Extension Package Interface, which allows the integration of new operators (implemented in C) in the form of so-called *extension packages*. The Extension Package Interface contains several predefined routines and macros for the easy handling of image data and memory objects in C. Once a new operator has been successfully integrated, it can be used like any other HALCON operator. The Extension Package Programmer's Manual contains detailed information about extending the operator library.

2.3 Language Interfaces

As shown in figure 2.1 on page 9, HALCON provides so-called *language interfaces*. These are native language bindings, that enable you to call operators and use HALCON data types directly from within your application, be it Python, C, C++, or .NET.

To start the development, we recommend to first check one of the ready-to-run example programs. Here, you can see how the project must be set up and how operators and types are used.

For each language interface, the names of types, classes, the naming conventions of operators, etc. may differ to be compliant with the typical rules that apply for the selected language. The operator signatures for the supported programming languages are documented in the HALCON Operator Reference.

2.3.1 HALCON/Python

The Python interface stands out for its simplicity and its ability for rapid prototyping. HALCON operators are called directly as standalone functions, after importing the HALCON/Python module. Note also that operator parameters in HALCON/Python are split into function parameters (inputs) and return values (output).

Example

The following code reads an image and computes the number of connected regions (page 10) in it.

```
img = ha.read_image('pcb')

region = ha.threshold(img, 0, 122)
num_regions = ha.count_obj(ha.connection(region))

print(f'Number of Regions: {num_regions}')
```

For prerequisites and a detailed walk-through, please see Programmer's Guide, Part 4, A First Example.

2.3.2 HALCON/C

The C interface is the simplest interface supported by HALCON. Each operator is represented by either one or two global functions where the operator name and the parameter sequence are identical to the HDevelop language.

Example

The following code reads an image and computes the number of connected regions (page 10) in it.

```
Hobject img;
read_image(&img, "pcb");

Hobject region;
threshold(img, &region, 0, 122);

Hobject connected_regions;
connection(region, &connected_regions);

Hlong num_regions = 0;
count_obj(connected_regions, &num_regions);

printf("Number of Regions: %" PRIdPTR "\n", num_regions);
```

For prerequisites and a detailed walk-through, please see Programmer's Guide, Part 5, A First Example.

2.3.3 HALCON/C++

The C++ interface is much more sophisticated than the C interface. Here, the advantages of C++ and object-oriented programming are used, i.e., automatic type conversion, construction and destruction, or grouping functions together with their data into classes. Like in the C interface, global functions for each HALCON operator are provided for a procedural style of programming.

Example

The following code reads an image and computes the number of connected regions (page 10) in it.

```
HImage img{"pcb"};

HRegion region = img.Threshold(0, 122);
Hlong numRegions = region.Connection().CountObj();

std::cout << "Number of Regions: " << numRegions << '\n';
```

For prerequisites and a detailed walk-through, please see Programmer's Guide, Part 2, A First Example.

2.3.4 HALCON/.NET

C# and Visual Basic.NET use HALCON via the .NET interface.

Analogously to C++, two styles of programming are offered: procedural and object-oriented. For the procedural style, the class `HOperatorSet` provides all HALCON operators, where `HObject` is used to handle iconic data and `HTuple` is used for control data. For the object-oriented style, classes like `HDataCode2d`, `HMeasure`, or `HShapeModel` are provided for the central functionality. In addition, classes for iconic data, e.g., `HImage` or `HRegion`, are available.

Example

The following code reads an image and computes the number of connected regions (page 10) in it.

```
HImage img = new HImage("pcb");

HRegion region = img.Threshold(0d, 122d);
int numRegions = region.Connection().CountObj();

Console.WriteLine("Number of Regions: " + numRegions);
```

For prerequisites and a detailed walk-through, please see Programmer's Guide, Part 3, A First Example.

2.4 Image Acquisition Interfaces

HALCON’s image acquisition interfaces form the bridge between software provided by the manufacturer of the image acquisition device and HALCON. They form a common, generic interface that requires a small set of operators only. Please refer to the Solution Guide II-A for detailed information about this topic.

Currently, HALCON provides interfaces for more than 50 frame grabbers and hundreds of industrial cameras in the form of dynamically loadable libraries (Windows: DLLs; Unix-like systems: shared libraries). Library names start with the prefix `hAcq`; the libraries ending with the suffix `xl` are used by HALCON XL.

The most widely used interfaces based on industry standards are already installed together with the HALCON libraries. Additional interfaces, as well as the latest versions of already included interfaces can be downloaded under `http://www.mvtec.com/products/interfaces`. The HALCON image acquisition interfaces may change more frequently than the HALCON library itself. One reason for this is that MVTec continuously develops new interfaces; furthermore, if the software provided by the manufacturers of image acquisition devices changes, e.g., if new features are integrated, the corresponding HALCON interfaces will be adapted. Please also refer to the Image Acquisition Interface Reference for a full list of supported image acquisition interfaces.

Once you successfully installed your image acquisition device, all you need to do to access it from HALCON is to call the operator `open_framegrabber`, specifying the name of the image acquisition interface and some additional information, e.g., regarding the connected camera. Then, images can be grabbed by calling the operator `grab_image` (or `grab_image_async`).

2.5 I/O Interfaces

HALCON provides interfaces for several I/O devices to enable data acquisition. These interfaces are available as dynamically loadable libraries (Windows: DLLs; Unix-like systems: shared libraries). Library names start with the prefix `hio`; the libraries ending with the suffix `xl` are used by HALCON XL.

The HALCON I/O device interfaces provide unified access to different I/O devices using a small set of operators. After you have installed your I/O device, a connection is established using the operator `open_io_device`, specifying the name of the I/O device interface and, optionally, some device-specific parameters. Once the connection is established, a transmission channel can be opened by calling `open_io_channel`. To read and write values on this channel, use the operators `read_io_channel` and `write_io_channel`, respectively.

Please note that the HALCON I/O device interfaces may change more frequently than the HALCON library itself. You can find the latest information together with downloadable interfaces (including documentation) under `http://www.mvtec.com/products/interfaces`. Please also refer to the I/O Device Interface Reference for a full list of supported I/O device interfaces.

Chapter 3

How to Develop Applications

We recommend that you start with rapid prototyping in HDevelop, the interactive development environment for the HALCON machine vision library. You can use HDevelop to find the optimal operators and parameters to solve your image analysis task. After developing an HDevelop program according to the given requirements, it has to be translated into its final environment. For this, you can choose between the following three approaches, depending on your preferences:

- **Start from Scratch:** Writing your program from scratch means to translate your HDevelop code into the target programming language (C++, Python...) manually. As mentioned before, the naming conventions of operators, the names of classes, etc., may differ between programming languages. Have a look at the HALCON Operator Reference to get the HALCON operator signatures for each supported programming language. For information on how to create applications in your desired target language, please read the Programmer's Guide.
- **Export HDevelop Code:** Translate your HDevelop code into the target programming language automatically using HDevelop's code export.
- **Export Library Project:** HDevelop's library export generates a ready-to-use project folder, including wrapper code in the target language and the CMake file to build the project. HDevelop's library export uses the HDevEngine, a library that acts as an interpreter. HDevEngine lets you directly execute HDevelop programs or procedures from an application written in C++ or any language that can integrate .NET objects. Thus, you do not have to recompile the entire application when making changes to the HDevelop code.

  Of course, you can use the HDevEngine without using HDevelop's library export function. How to use HDevEngine is described in detail in the Programmer's Guide, Part 6 (Using HDevEngine).

3.1 HDevelop

Let's take a first look at HDevelop. Figure 3.1 shows HDevelop's user interface, after a program has been loaded and partly been executed.

By default, these windows are visible, which are also essential for developing with HDevelop:

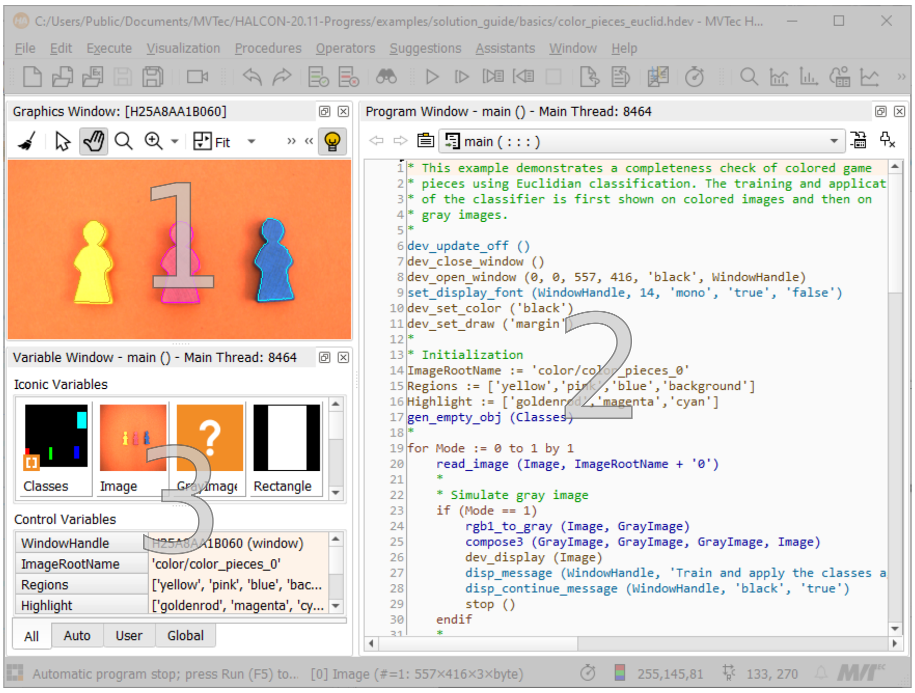
(1) Graphics Window Displays (intermediate) results, namely iconic data (page 10) like images, regions, and XLDs.

(2) Program Window This is where you type your program code, using operators (page 10) to access HALCON's image processing methods.

(3) Variable Window Shows all variables, namely iconic variables and control variables. Iconic variables contain iconic data (page 10) and control variables contain control data (page 10).

Detailed information about HDevelop can be found in the HDevelop User's Guide. Our tutorial videos also offer a good introduction to HDevelop:

HDevelop Tutorial 01: GUI and Navigation
HDevelop Tutorial 02: Variables
HDevelop Tutorial 03: Visualization


C:/Users/Public/Documents/MVTec/HALCON-20.11-Progress/examples/solution_guide/basics/color_pieces_euclid.hdev - MVTec H...
File Edit Execute Visualization Procedures Operators Suggestions Assistants Window Help
Graphics Window: [H25A8AA1B060]
Fit
1
2
3
Variable Window - main () - Main Thread: 8464
Iconic Variables
Classes
Image
GrayImage
Rectangle
Control Variables
All
Auto
User
Global
Program Window - main () - Main Thread: 8464
main (: : :)
Automatic program stop; press Run (F5) to...
[0] Image (#=1: 557×416×3×byte)
255,145,81
133, 270

WindowHandle	H25A8AA1B060 (window)
ImageRootName	'color/color_pieces_0'
Regions	['yellow', 'pink', 'blue', 'bac...
Highlight	['goldenrod', 'magenta', 'cy...

```
* This example demonstrates a completeness check of colored game
* pieces using Euclidian classification. The training and applicat
* of the classifier is first shown on colored images and then on
* gray images.
*
dev_update_off ()
dev_close_window ()
dev_open_window (0, 0, 557, 416, 'black', WindowHandle)
set_display_font (WindowHandle, 14, 'mono', 'true', 'false')
dev_set_color ('black')
dev_set_draw ('margin')
*
* Initialization
ImageRootName := 'color/color_pieces_0'
Regions := ['yellow','pink','blue','background']
Highlight := ['goldenrod','magenta','cyan']
gen_empty_obj (Classes)
*
for Mode := 0 to 1 by 1
    read_image (Image, ImageRootName + '0')
    *
    * Simulate gray image
    if (Mode == 1)
        rgb1_to_gray (Image, GrayImage)
        compose3 (GrayImage, GrayImage, GrayImage, Image)
        dev_display (Image)
        disp_message (WindowHandle, 'Train and apply the classes a
        disp_continue_message (WindowHandle, 'black', 'true')
        stop ()
    endif
    *
```

Figure 3.1: HDevelop's User Interface.

3.2 Example Program

Now that you have been introduced to HDevelop's User Interface and the basic concepts of HALCON (page 9), let's develop a C++ application using the Library Export Approach.

In addition to the following step-by-step instructions, we recommend that you watch our tutorial videos:

Integrate HDevelop code into a C++ application using the Library Project Export

Integrate HDevelop code into a C# application using the Library Project Export

The videos demonstrate the library export and provide more background information about the topic.

3.2.1 Create Prototype in HDevelop

The task of the following example is to read an image and count the number of connected regions it it.

1. Open HDevelop and enter the following code into the Program Window:

```
read_image (Image, 'pcb')
threshold (Image, Region, 0, 122)
connection (Region, ConnectedRegions)
count_obj (ConnectedRegions, Number)
```

2. Test your program by clicking Run in the toolbar or pressing F5.

To easily integrate this HDevelop code into an actual application, we encapsulate the machine vision part in a local procedure.

1. Highlight the following code lines:

```
threshold (Image, Region, 0, 122)
connection (Region, ConnectedRegions)
count_obj (ConnectedRegions, Number)
```

2. Right-click to open the context menu.

3. Choose Create New Procedure.
4. Name it count_regions.
5. Select Parameters and change Selection Scheme to First In, Last Out.
6. Confirm with OK.
7. Save your HDevelop program as hdev_count_regions.hdev.

3.2.2 Prepare Visual Studio Project

In this example, we will use Visual Studio 2019. [1]

1. Create an empty C++ Windows Console project and name it vs_count_regions. Please activate the option Place solution and project in the same directory. [2]
2. Add a C++ source file (Menu Project ▷ Add New Item... ▷ C++ File) and name it vs_count_regions.cpp.
3. Choose the solution platform x64 from the drop down menu in the toolbar.
4. Open your project properties (Menu Project ▷ vs_count_regions Properties...) and make the following settings:
    - Select C/C++ ▷ General and add the following Additional Include Directories:

      $(HALCONROOT)\include;$(HALCONROOT)\include\halconcpp;
    - Select Linker ▷ General and add the following Additional Library Directory:

      $(HALCONROOT)\lib\$(HALCONARCH);
    - Select Linker ▷ Input and add the following Additional Dependencies:

      halconcpp.lib;hdevenginecpp.lib;

3.2.3 Export Library Project

Next, we export our HDevelop program hdev_count_regions.hdev into our Visual Studio project folder.

1. Open the previously created HDevelop program hdev_count_regions.hdev.
2. Open File ▷ Export Library Project...
3. Make the following settings:
    - Input file: Current Program
    - Target Language: C++
    - Project Name: hdev_count_regions
    - Project Location: Choose the location of our Visual Studio project vs_count_regions.
    - Namespace: hdev_count_regions
4. Confirm with Export.

[1] We recommend using the latest supported version of Visual Studio.
[2] The directory structure is important, because the final C++ program includes relative paths.

Now, your Visual Studio project folder `vs_count_regions` should contain at least the following data:

▷ `vs_count_regions.cpp` (Source File)
▷ `vs_count_regions.sln` (Solution)
▷ `hdev_count_regions` (Folder from HDevelop Export)
  ▷ `cmake`
  ▷ `res_hdev_count_regions`
    ▷ `hdev_count_regions.hdev`
  ▷ `source`
    ▷ `hdev_count_regions.cpp`
    ▷ `hdev_count_regions.h`
  ▷ `CMakeLists.txt`

3.2.4 Integrate Library Project Into Visual Studio

Lastly, we have to integrate the HDevelop program into our Visual Studio Project.

1. Open the Visual Studio Project.
2. Open `Project` ▷ `Add Existing Item...` and choose the C++ file `hdev_count_regions.cpp` and the header file `hdev_count_regions.h`, created by HDevelop's Library Export. (The files are located in the folder `hdev_count_regions` ▷ `source`.)
3. Enter the following code into `vs_count_regions.cpp`:

```
#include <iostream>

#include "HalconCpp.h"
#include "hdev_count_regions/source/hdev_count_regions.h"

int main()
{
  HalconCpp::HImage Image("pcb");

  hdev_count_regions::SetResourcePath("hdev_count_regions/res_hdev_count_regions");

  HalconCpp::HTuple Number{};
  hdev_count_regions::count_regions(Image, &Number);

  std::cout << "Number of Regions: " << Number.L() << '\n';
}
```

4. Execute the program. → A console opens, showing the result 'Number of regions: 43'.

Chapter 4

How to Continue

To dive deeper into HALCON, we offer further documentation and support.

- **MVTec Academy**
  The MVTec Academy offers a big variety of interactive online courses, covering many topics, from basic concepts to advanced application-specific trainings. Visit `https://academy.mvtec.com` to find out more.

- **HDevelop Example Programs**
  HALCON provides an extensive set of example programs, not only for HDevelop but also for different programming languages. These examples can be found in the directory denoted by the environment variable %HALCONEXAMPLES% or, if the variable is not set, in the subdirectory examples of the folder into which you have installed HALCON.

  To open an HDevelop example program, select the menu `File` ▷ `Browse HDevelop Example Programs...`.
  To learn about some basic HALCON concepts we recommend to have a look at the example program `halcon_basic_concepts.hdev`.

- **Services and Support**
  Our website `http://www.mvtec.com/services-support` offers a variety of support, for example tutorial videos, information about workshops and trainings, the developers' corner providing tips and tricks, and many more.

- **HALCON Documentation**
  The documentation provides a wealth of information, from beginner topics to expert knowledge. For example, our Solution Guides describe machine vision methods and how to apply them in HDevelop. A good starting point is Solution Guide I which introduces you to the main machine vision methods.

  An overview of all manuals with a short description can be found on the documentation's entry page.